REMARKS

The Examiner has rejected claims 1-18. Applicant has amended claims 1, 2, 4, 5, and 7. Applicant has added new claims 19-24. Reexamination and reconsideration of pending claims 1-24 is respectfully requested.

OBJECTION TO THE SPECIFICATION BASED ON 35 U.S.C. § 112, FIRST PARAGRAPH

The Examiner has rejected the specification under 35 U.S.C. §112, first paragraph, as failing to adequately teach how to make and/or use the invention. The Examiner states that it is unclear how a receiver object determines whether it has been given all the information it needs to execute a message to determine whether a query must be generated and sent back to the sender object. The Examiner states that there must be "an explanation of how this 'query' feature is accomplished, i.e., when a query is done, under what circumstances, based on what factors, how is it implemented." (emphasis in original) The Examiner states:

> "[i]t is common to expect that if an object receives an incomplete method call, that it would produce an error message. However, this is far from the described capability of an object to determine whether additional information is needed for execution, to generate a 'query' if more information is needed, and for the sender to have the functionality to understand this 'query' and respond with a 'reply'."

Applicant contends that the specification is sufficient to enable one skilled in the art to make and use the invention, and, in particular, the 'query' feature noted by the Examiner.

Applicant respectfully refers the Examiner to the specification starting at page 23, line 6, for example. As indicated in the specification, when a receiver object requires additional information from a sender object, it generates a request message. The sender object receives the request and generates a response which is sent to the receiver object.

Further, the specification, starting at page 23, line 25, provides an example of the recursive nature of the present invention where a remote object does not recognize a method. When a remote object does not recognize a method (e.g., "*foo*") sent to it by a local object, for example, the remote object asks the sending object "*what is foo*?" All objects recognize the method "*what is*". The sender object can then respond with instructions concerning the nature of the method being investigated. As illustrated in the specification (starting at page 23, line 30), the sender object might reply that "*foo*" is a method that requires an integer. Once the receiver object receives the instruction (e.g., a message that "*foo*" is a method that requires an integer), the receiver object can determine whether or not an integer has been provided. If no integer has been provided, the remote object can then generate a second request to get the integer. The process for obtaining the integer would be similar.

The above example discloses the capability of an object to determine whether additional information is needed, to generate a 'query' to get the additional information, to understand this 'query' and respond to this 'query'. That is, the remote object determines that it does not understand "*foo*". As a result, the remote object generates a "*what is foo?*" message. Since all objects understand the "*what is foo?*" message, the sender is able to understand, and respond to, the query initiated by the remote object. The sender responds by providing a further explanation of "*foo*". In the example, this explanation includes the fact that "*foo*" requires an integer. Based on the reply from the sender, the remote object knows that "*foo*" needs an integer. The remote object can then determine whether it received an integer argument as part of the original message. If it did not receive an integer as part of the original message, the remote object (as indicated in the example) can generate another message to request the integer argument.

Therefore, as indicated in the specification, the remote object determines that it needs additional information for execution (e.g., additional

information regarding "*foo*") and it generates a 'query' (e.g., "*what is foo?*") to obtain additional information. Since "what is" is global, the sender has the functionality to understand the 'query'. Further, the sender is able to respond to the 'query'. The sender replies by providing additional information about "*foo*".

Applicant therefore contends that the Specification is sufficient to enable one skilled in the art to make and use the invention, and, in particular, the 'query' feature.

REJECTION OF CLAIMS 5-7 AND 11-18 BASED ON 35 U.S.C. § 112, FIRST PARAGRAPH

The Examiner rejects Claims 5-7 and 11-18 under 35 U.S.C. § 112, first paragraph, for the reasons set forth in the objection to the Specification. Applicant respectfully refers the Examiner to the Applicant's response to the 35 U.S.C. § 112, first paragraph rejection provided above. Given the response provided above, Applicant contends that Claims 5-7 and 11-18 are allowable.

REJECTION OF CLAIMS 1-4 AND 8-10 BASED ON 35 U.S.C §§ 102(a) & (b)

The Examiner rejects Claims 1-4 and 8-10 under 35 U.S.C. §§ 102(a) and (b) as being unpatentable over Bennett, "The Design and Implementation of Distributed Smalltalk", OOPSLA '87 Proceedings: Conference on Object Oriented Programming, Systems, Languages, and Applications, pp. 318-30, 12/1987 (hereinafter referred to as Bennett).

The Examiner states that Bennett indicate a system-dependent form of the message. The Examiner states that Bennett teaches a method that uses the **doesNotUnderstand:**, **perform:**, and **remoteSend:** primitives, a "RemoteObjectTable" which uses a "messageProcess" to construct a "messageArray" and the encoding of an "argument string". The Examiner

states that these features indicate a system-dependent form which is used during transmission of the message.

The Examiner states that Applicants' arguments filed August 23, 1994 and relating to Bennett were directed to perceived differences between the invention and the cited reference rather than being directed to delineating specifically claimed features of the invention not taught by the cited references. The Examiner states that Claim 1 describes the transmission of a message to a first proxy in said first process. The Examiner states that Bennett clearly describes ProxyObjects present in each host, this host inherently being a process address space.

Applicants respectfully disagree. Applicants contend that Claim 1 differs from Bennett for at least the following reasons:

1. In Bennett, a host is a computer or machine. A process address space is not the same as a host.

2. In Bennett, ProxyObject and RemoteObjectTable are centralized processes that are not included in either the sending or receiving processes.

1. <u>In Bennett, a host is a computer or machine. A process address space is not the same as a host.</u>

The Examiner states that:

> "Bennett clearly describes ProxyObjects present in each host, this host inherently being a process address space.

Applicants respectfully disagree. Bennett uses the terms "host" and "machine" interchangeable or in combination to mean the same thing (i.e., a

computer having a processor and memory). On page 318, left-hand column, Bennett states:

> "[a]lthough Smalltalk host machines can be interconnected with high bandwidth networks, only rudimentary support exists within Smalltalk for cooperation among users, and no support exists within Smalltalk for object sharing between users, communication between objects that reside on different machines, or cooperation between processes on several machines." (emphasis added.)

Thus, Bennett combines the terms "host" and "machine" in the previous passage. In addition, Bennett uses the terms "host" and "machine" interchangeably. For example, in discussing class replication in the Distributed Smalltalk design in Bennett, option (1) on page 320, left-hand column is chosen. Option (1) states:

> "[d]isallow remote classes (i.e., require that classes and instances be co-resident.) This approach impacts object mobility. Instances can only move to hosts with compatible classes. Ensuring class compatibility is hard." (emphasis added.) (page 320, left-hand column)

In further discussion of Option (1), Bennett states (at page 320, left-hand column):

> "[c]lass replication in a reactive system can also create problems. If users on different machines are allowed to create incompatible versions of a system class, considerable confusion can ensue if instances of these classes move between machines." (emphasis added.) (page 320, left-hand column)

The following quotes from Bennett further illustrate that the terms "host" and "machine" are used interchangeably in Bennett:

1. ProxyObjects' response to the *doesNotUnderstand:* message is to forward the original message to the RemoteObjectTable on the appropriate machine. (emphasis added.) (page 323, left-hand column.)

2. There is one RemoteObjectTable per host. (emphasis added.) (page 323, right-hand column.)

Therefore, in Bennett, the terms "host" and "machine" are used interchangeably to mean the same thing. In Bennett, both of the terms refer to a computer that has a processor, memory, etc.

The Examiner states that, in Bennett, a host is inherently a "process address space." Applicants respectfully disagree. As indicated above, in Bennett, a host or machine is a computer having a processor and memory. A process address space is the portion of memory that is addressable by a process executing in a computer system. In Bennett, a host or machine is different than a process. In Bennett, a process executes in a host. In Bennett, a RemoteObjectTable executes on a host. As stated in Bennett:

> "The RemoteObjectTable has associated with it three processes: the messageProcess, the userProcess, and the kernelProcess. The messageProcess waits for any Distributed Smalltalk message to the host. Upon receipt of such a message, the messageProcess constructs a messageArray containing the remoteSend parameters." (page 324, left-hand column)

Thus, in Bennett, the RemoteObjectTable for a host has three processes that are executing on the host. Each process has a range of memory (or address space) that the process can access. Therefore, a host has at least three process address spaces for each of these processes. Further, as stated in Bennett (page 319, left-hand column), each user has an address space. Thus, in Bennett a host is not a process address space. Rather, a host has multiple address spaces to accommodate, for example, each user and the processes associated with the RemoteObjectTable.

2. <u>In Bennett, ProxyObject and RemoteObjectTable are centralized processes that are not included in either the sending or receiving processes.</u>

In Bennett, a ProxyObject is centralized on a host or machine. In Bennett, a sending process must access the centralized ProxyObject process to forward a message to a remote object on a remote host.

In Bennett, the remote host (remote machine) has a centralized RemoteObjectTable that is responsible for receiving and replying to messages forwarded by the ProxyObject. The receiver process is scheduled by the RemoteObjectTable when a message is received from a proxyObject. The receiver process executes the required action and returns a value to the centralized RemoteObjectTable for transmittal to the ProxyObject. As stated in Bennett:

"The RemoteObjectTable is responsible for receiving and replying to messages forwarded by proxyObjects. There is one RemoteObjectTable per host. It is the sole instance of class RemoteObjectTable .... When the RemoteObjectTable receives a message from some proxyObject, it schedules a process that will contain the execution context of the actual message receiver by sending the message perform to the receiver with the forwarded selector and argument (if any) as arguments to the perform message." (page 323, right-hand column)

In Bennett, therefore, the ProxyObject is not in the sending process. Further, the process that executes the message is not the same process as the RemoteObjectTable (i.e., the receiving process). Therefore, in Bennett, the sending process, receiving process, ProxyObject process, RemoteObjectTable process are different processes. In contrast, in claim 1 of the present application the first proxy object is contained in the sending process. Further, in claim 1 of the present application, the process that executes the message is the same as the receiving process.

Therefore, Applicants contend that claim 1 is allowable. Further, Applicants contend that claims 2-4 and 8-10 being dependent on an allowable dependent claim are, therefore, allowable.

For the foregoing reasons, applicant contends that none of the references cited, either alone or in combination, teach, describe, or suggest the present invention. Applicant contends that pending claims 1-24 are allowable.

Respectfully submitted,

Hecker & Harriman

Date: 2/22/95

By: _____
J. D. Harriman II
Reg. No. 31,0967

2029 Century Park East
Suite 1600
Los Angeles, CA 90067
(310) 286-0377

File No.: 10010.929C

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231, on February 22, 1995

2/22/95 Rhonda Royba
Signature Reg. No. Date